3 Pulse Waveform

3.1 Overview

The class ***PulWaveform*** embodies a pulse waveform in NMR simulations. The class maintains the pulse waveform as a discrete function whose individual points (waveform steps) consist of three values, $\{\gamma B_1, \phi, t_p\}$, the rf-field strength, the rf-field phase, and the time the rf-field is applied. This class works with other GAMMA classes for handling shaped pulses, composite pulses, & pulse cycles.

Waveform in GAMMA Pulse Hierarchy

Waveform **Composite/Shaped Pulse** **Pulse Cycle**

step	γB_1	ϕ	time
1	2.5	0	0.1
2	2.5	90	0.2
3	2.5	0	0.1

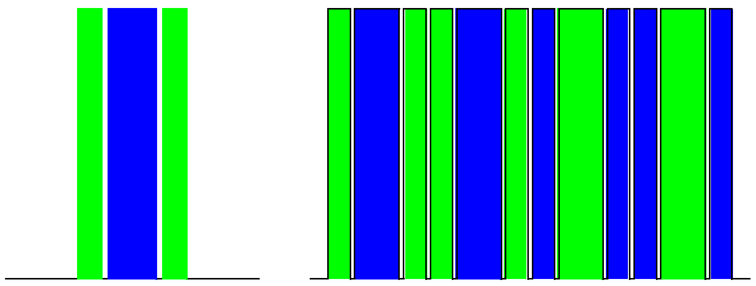

Figure 3-1 A waveform (e.g. composite 180) is just a list of pulse steps, each step having a specified rf-field strength, rf-field phase, and length (e.g. shown in kHz and msec). The waveform is used to produce a shaped or composite pulse which can be used as a step in an NMR pulse sequence. In turn, the composite pulse can be used to form a pulse cycle (repeated pulses with phase changes, e.g. MLEV-4) which may also be used during NMR simulations. This hierarchy continues to accomodate supercycles and cycles with interdispersed pulses.

3.2 Chapter Contents

3.2.1 Pulse Waveform Functions

3.2.2 Pulse Waveform Sections

3.2.3 Pulse Waveform Figures & Tables

3.3 Construction & Assignment

3.3.1 PulWaveform

Usage:

```
#include <PulWaveform.h>
PulWaveform()
PulWaveform(row_vector& wfsteps, row_vector& wftimes, const String& wfname, int wfrad=0)
PulWaveform(const PulWaveform& PWF)
```

Description:

The function ***PulWaveform*** is used to create a pulse waveform in GAMMA.

1. PulWaveform() - Creates an “empty” NULL pulse waveform. Can be later filled by an assignment.
2. PulWaveform(row_vector& wfsteps, row_vector& wftimes, const String& wfname, int wfrad=0) - Creates a new waveform named ***wfname*** whose step rf-amplitudes and phase are contained in the input vector ***wfsteps*** and whose step lengths are in the vector wftimes. The flag ***wfrad*** indicates whether the input phases are in degrees (default) or radians.
3. PulWaveform(const PulWaveform& PWF) - Called with another pulse waveform quantity this function constructs an identical waveform equal to the input ***PWF***.

Return Value:

PulWaveform returns no parameters. It is used strictly to create an pulse waveform.

Examples:

```
#include <PulWaveform.h>
PulWaveform PWF;                  // Empty pulse waveform.
int nsteps = 100;
row_vector gB1s = Lorentzian(
row_vector times(nsteps, 0.001);
PWF1(gB1s, times, “new”);         // Here is a mock Lorentzian pulse waveform
PulWaveform PWF2(PWF1);           // An identical copy of PWF1
```

See Also: =

3.3.2 =

Usage:

```
#include <PulWaveform.h>
void PulWaveform operator = (PulWaveform &PWF)
```

Description:

The unary ***operator*** **=** (the assignment operator) allows for the setting of one pulse waveform equal to another pulse waveform. The waveform being assigned to will be overwritten by ***PWF***.

Return Value:

None, the function is void

Examples:

```
#include <PulWaveform.h>
#include <PulWALTZ.h>
PulWaveform PWF;                    // Empty pulse waveform.
PWF = WF_WALTZQ(875.0);             // Set PWF to WALTZ-Q waveform @ 875 Hz
```

See Also: PulWaveform

3.4 Access Functions

3.4.1 steps

Usage:

```
#include <PulWaveform.h>
int PulWaveform::steps( )
double PulWaveform::steps(double td)
```

Description:

The function ***steps*** with no arguments returns the number of individual steps defined in the pulse waveform. Of a length ***td*** is given, the function returns the number of steps that will span the length specified.

Return Value:

The function returns either an integer or a double.

Example:

```
#include <PulWaveform.h>
#include <PulMLEV.h>
MLEV MP(1260.0, “1H”);                 // Set up MLEV parameters
PulWaveform PWF = MP.WF();             // Set default MLEV waveform (comp. 180)
cout << “\nMLEV Waveform has “         // Output the number of steps
     << PWF.steps() << “ steps.”;
double td = 0.03;                      // Set a delay time
cout << “\nTo span “ << td << “ seconds “   // Output waveforms spanning delay
     << “requires “ << PWF.steps(td)
     << “ waveforms”;
```

See Also:

3.4.2 name

Usage:

```
#include <PulWaveform.h>
String PulWaveform::name( )
```

Description:

The function ***name*** returns the name of the pulse waveform.

Return Value:

The function returns a string.

Example:

```
#include <PulWaveform.h>
#include <PulGARP.h>
GARP GP(1260.0, “1H”);                     // Set up GARP parameters
PulWaveform PWF = GP.WF();                 // Set default GARP waveform (25 steps)
cout << “\nWorking with “ << PWF.name()    // Output the waveform name
      << “ waveform.”;
```

3.4.3 values

Usage:

```
#include <PulWaveform.h>
row_vector PulWaveform::values( )
```

Description:

The function ***values*** returns a row_vector containing values which define the pulse waveform steps. The ith vector value contains the values {γB1, ϕ}, where the real component is the rf-field strength in Hz, and the imaginary component is the rf-phase in degrees (or radians).

Return Value:

The function returns a row vector.

Example:

```
#include <PulWaveform.h>
#include <PulGARP.h>
GARP GP(1260.0, “1H”);                     // Set up GARP parameters
PulWaveform PWF = GP.WF();                 // Set default GARP waveform (25 steps)
row_vector vals = PWF.values();            // Get array of strengths and phases
```

See Also:

3.4.4 lengths

Usage:

```
#include <PulWaveform.h>
row_vector PulWaveform::lengths( )
```

Description:

The function ***lengths*** returns a row_vector containing values which define the pulse waveform step lengths. The real part of the ith vector value contains the length os step i in seconds.

Return Value:

The function returns a row vector.

Example:

```
#include <PulWaveform.h>
#include <PulGARP.h>
GARP GP(1260.0, “1H”);                // Set up GARP parameters
PulWaveform PWF = GP.WF();            // Set default GARP waveform (25 steps)
row_vector ts = PWF.lengths();        // Get array of lengths
```

See Also:

3.4.5 value

Usage:

```
#include <PulWaveform.h>
complex PulWaveform::value(int i)
```

Description:

The function ***value*** returns a compex number for the values which define the pulse waveform step ***i***. The value contains the number {γB1, ϕ}, where the real component is the rf-field strength in Hz, and the imaginary component is the rf-phase in degrees (or radians).

Return Value:

The function returns a complex number.

Example:

```
#include <PulWaveform.h>
```

See Also:

3.4.6 phase

Usage:

```
#include <PulWaveform.h>
double PulWaveform::phase(int i)
```

Description:

The function ***phase*** returns the value of the rf-field phase at pulse waveform step ***i*** in degrees (or radians).

Return Value:

The function returns a double.

3.4.7 strength

Usage:

```
#include <PulWaveform.h>
double PulWaveform::strength(int i)
```

Description:

The function ***strength*** returns the value of the rf-field amplitude at pulse waveform step ***i*** in Hz.

Return Value:

The function returns a double.

Example:

```
#include <PulWaveform.h>
```

See Also:

3.4.8 length

Usage:

```
#include <PulWaveform.h>
double PulWaveform::length(int i)
```

Description:

The function ***length*** returns the length of th pulse waveform in seconds.

Return Value:

The function returns a double.

Example:

```
#include <PulWaveform.h>
```

See Also:

3.5 Auxiliary Functions

3.5.1 maxlength

Usage:

```
#include <PulWaveform.h>
double PulWaveform::maxlength( )
```

Description:

The function ***maxlength*** returns the length of the longest waveform step.

Return Value:

The function returns a double.

Example:

```
#include <PulWaveform.h>
```

See Also:

3.5.2 minlength

Usage:

```
#include <PulWaveform.h>
double PulWaveform::maxlength( double cutoff=1.e-13)
```

Description:

The function ***minlength*** returns the length of the shortest non-zero waveform step. The step is considered to be of zero length if it falls below the value set by ***cutoff***.

Return Value:

The function returns a double.

Example:

```
#include <PulWaveform.h>
```

See Also:

3.5.3 gamB1const

Usage:

```
#include <PulWaveform.h>
int PulWaveform::gamB1const( )
```

Description:

The function ***gamB1const*** returns true if all steps in the waveform have the same rf-field strength.

Return Value:

The function returns an integer.

Example:

```
#include <PulWaveform.h>
```

See Also: phaseconst, timeconst

3.5.4 phaseconst

Usage:

```
#include <PulWaveform.h>
int PulWaveform::phaseconst( )
```

Description:

The function ***phaseconst*** returns true if all steps in the waveform have the same rf-field phase.

Return Value:

The function returns a string.

Example:

See Also: timeconst, gamB1const

3.5.5 timeconst

Usage:

```
#include <PulWaveform.h>
int PulWaveform::timeconst( )
```

Description:

The function ***timeconst*** returns true if all steps in the waveform have the same length.

Return Value:

The function returns a row vector.

Example:

```
#include <PulWaveform.h>
```

See Also: phaseconst, gamB1const

3.5.6 WFs

Usage:

```
#include <PulWaveform.h>
double PulWaveform::WFs(double td)
```

Description:

The function ***WFs*** returns the number of pulse waveforms needed to span the length ***td***.

Return Value:

The function returns a double.

Example:

```
#include <PulWaveform.h>
```

See Also:

3.5.7 fullWFs

Usage:

```
#include <PulWaveform.h>
int PulWaveform::fullWFs(double td)
```

Description:

The function ***fullWFs*** returns the number of complete pulse waveforms that fit within the time span ***td***.

Return Value:

The function returns an integer.

Example:

```
#include <PulWaveform.h>
```

See Also:

3.5.8 fullsteps

Usage:

```
#include <PulWaveform.h>
int PulWaveform::fullsteps(double td)
```

Description:

The function ***fullsteps*** returns the number of complete pulse waveform steps that fit within the time span ***td***.

Return Value:

The function returns an integer.

Example:

```
#include <PulWaveform.h>
```

3.5.9 sumlength

Usage:

```
#include <PulWaveform.h>
double PulWaveform::sumlength(int i)
```

Description:

The function ***sumlength*** returns the summed length over the first ***i*** steps of the waveform.

Return Value:

The function returns a double.

Example:

```
#include <PulWaveform.h>
```

3.6 Plotting Functions

3.6.1 GP

Usage:

```
#include <PulWaveform.h>
void PulWaveform::GP(int type=0, int split=0, int ends=0, int N=1)
```

Description:

The function ***GP*** will produce a plot of the waveform on screen using the ***Gnuplot*** program if available. The function will plot either the rf-intensity versus time (***type = 1***) or the rf-phase versus time (***type = 0***). Individual waveforms steps will be separated by ***split*** multiples of one tenth the first waveform step length. Ends will be drawn on the plot of length ends*length of first pulse step. There will be ***N*** waveforms plotted.

Return Value:

Void. Aplot is produced on screen if Gnuplot is available.

Example:

```
#include <PulWaveform.h>
#include <PulMLEV.h>
MLEV MP(1260.0, “1H”);                 // Set up MLEV parameters
PulWaveform PWF = MP.WF();             // Set default MLEV waveform (comp. 180)
PWF.GP(1, 1, 1);                       // Make strength vs. time plot
```

See Also: FM

3.6.2 FM

Usage:

```
#include <PulWaveform.h>
void PulWaveform::FM(int type=0, int split=0, int ends=0, int N=1)
```

Description:

The function ***FM*** will produce a plot of the waveform in FrameMaker MIF format. The function will plot either the rf-intensity versus time (***type = 1***) or the rf-phase versus time (***type = 0***). Individual waveforms steps will be separated by ***split*** multiples of one tenth the first waveform step length. Ends will be drawn on the plot of length ends*length of first pulse step. There will be ***N*** waveforms plotted. The output filename will reflect the name of the waveform, the type plotted, and have a mif suffix.

Return Value:

Example:

```
#include <PulWaveform.h>
#include <PulMLEV.h>
MLEV MP(1260.0, “1H”);                 // Set up MLEV parameters
PulWaveform PWF = MP.WF();             // Set default MLEV waveform (comp. 180)
```

```
        PWF.FM(1, 1, 1);                                // Make strength vs. time plot
```

See Also: GP

3.7 Input/Output Functions

3.7.1 printBase

Usage:

```
#include <PulWaveform.h>
ostr PulWaveform::printBase(ostream& ostr)
```

Description:

The function ***printBase*** will put basic information regarding the waveform into the output stream ***ostr*** given as an input argument

Return Value:

The function modifies the output stream and returns it.

Example:

```
#include <PulWaveform.h>
```

See Also:

3.7.2 printSteps

Usage:

```
#include <PulWaveform.h>
ostr PulWaveform::printSteps(ostream& ostr)
```

Description:

The function ***printSteps*** will put information regarding the pulse cycle individual step phases into the output stream ***ostr*** given as an input argument.

Return Value:

The function modifies the output stream and returns it.

Example:

```
#include <PulWaveform.h>
```

See Also:

3.7.3 print

Usage:

```
#include <PulWaveform.h>
ostr PulWaveform::print(ostream& ostr, int full=0)
```

Description:

The function ***print*** will put information regarding the pulse waveform into the output stream ***ostr*** given as an input argument. If the optional flag ***full*** has been set to non-zero, information regarding individual pulse waveform steps will also be added (which can be a lot data) added to the output stream.

Return Value:

The function modifies the output stream and returns it.

Example:

```
#include <PulWaveform.h>
```

See Also:

3.7.4 <<

Usage:

```
#include <PulWaveform.h>
ostream& operator << (ostream& ostr, PulWaveform& PWF)
```

Description:

The operator **<<** adds the pulse waveform specified as an argument ***PWF*** to the output stream ***ostr***.

Return Value:

None.

Example(s):

```
#include <PulWaveform.h>
```

See Also:

3.8 Description

3.8.1 Introduction

Class ***PulWaveform*** is designed to faclitate the use of generic shaped pulses, composite pulses, and pulse trains in GAMMA. There are a wide variety of such waveforms commonly used in modern NMR spectroscopy. In GAMMA, as in an NMR experiment, we should like to use pulses and pulse trains generated from arbitrary waveforms as individual steps in a general pulse sequence. This includes use in variable delays as part of multi-dimensional experiements and/or use in continuous pulse trains during acquisition steps.

3.8.2 Pulse Waveform Basis

We consider a pulse waveform as involving four basic features: 1.) The ***# steps***, 2.) The ***rf-field strength*** of each step, the ***rf-phase*** of each step, the ***length*** of each step: N, $\{\gamma B_1, \phi, t_p\}$. One example would be a Gaussian pulse waveform. In this case the Gaussian function is broken up into N steps each having the same phase and length but with varying field strength.

Gaussian Waveform

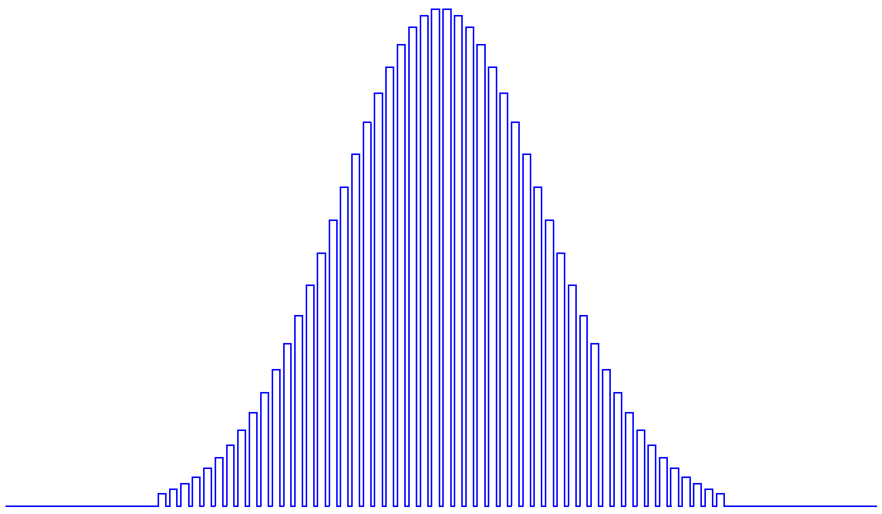

Figure 3-2 A Gaussian waveform in GAMMA. Each step here has the same length and phase whereas the rf intensity changes according the the Gaussian function.This figure was made by the program GaussWF.cc on page 55 of this chapter.

A Gaussian pulse waveform, such as shown above, simply maintains the steps and rf-amplitudes, in this case the step length is constant. As an alternative example, the we can consider a GARP-1 25-step composite pulse as our waveform. In this case the rf-field strength is maintained constant whereas the phase and length of the steps (individual pulses) change.

Basic GARP 25 Step Sequence

Step	Angle	Step	Angle	Step	Angle
1	30.5	9	134.5	17	258.4
2	$\overline{55.2}$	10	$\overline{256.1}$	18	$\overline{64.9}$
3	257.8	11	66.4	19	70.9
4	$\overline{268.3}$	12	$\overline{45.9}$	20	$\overline{77.2}$
5	69.3	13	25.5	21	98.2
6	$\overline{62.2}$	14	$\overline{72.7}$	22	$\overline{133.6}$
7	85.0	15	119.5	23	255.9
8	$\overline{91.8}$	16	$\overline{138.2}$	24	$\overline{65.6}$
				25	$\overline{53.4}$

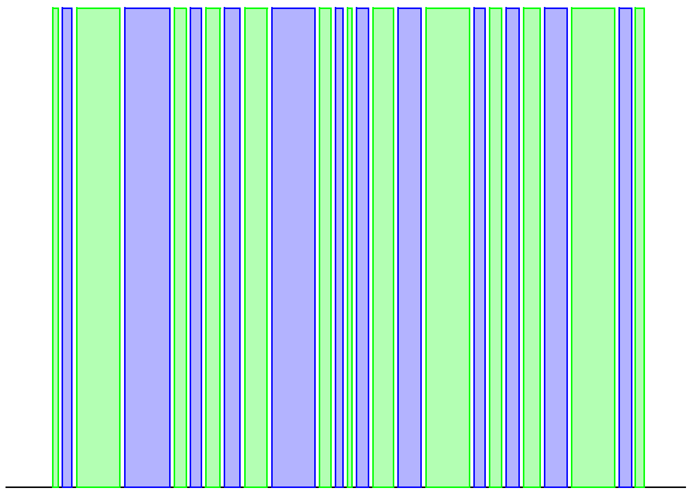

Figure 3-3 The basic 25-step GARP waveform. The blue steps indicate pulse that are applied with a 180 degree phase shift, as indicate by a bar in the table listing. The program which produced this plot can be found at the end of this chapter, GarpWF0.cc on page 55.

See the DANTE module for an example of a waveform with pulse and delay steps mixed and the CHIRP module should contain a nice example of a constantly changing (phase and intensity) waveform.

3.8.3 Pulse Waveform Construction

The examples in the previous section generated and plotted waveforms which are supplied through pre-existing functions available in GAMMA. Users are free to look through the source code of the individual modules containing the function(s) to see exactly how they were generated. In this section it is shown explicitly how you can build your own pulse waveforms.

Building a waveform requires two arrays (row-vectors) that have length equal to the intended waveform steps. The first vector will contain the rf strengths and phases and the second will contain the step lengths. An optional name can be associated with the waveform too. Let us just make the basic GARP 25-step waveform given in a previous figure. First, here is how to make the vector of amplitudes and phase:

```
row_vector WFsteps(25);                        // Vector for waveform
double gamB1 = 2000;                           // Field strength to 2 kHz
double phi = 0.0;                              // Base phase
double phibar = phi + 180.0;                   // Alternate phase
WFsteps.put(complex(gamB1,phi),      0);       // Set waveform values
WFsteps.put(complex(gamB1,phibar), 1);         //  { gamB1, phi }
WFsteps.put(complex(gamB1,phi),     2);
WFsteps.put(complex(gamB1,phibar), 3);
WFsteps.put(complex(gamB1,phi),     4);
WFsteps.put(complex(gamB1,phibar), 5);
WFsteps.put(complex(gamB1,phi),     6);
WFsteps.put(complex(gamB1,phibar), 7);
WFsteps.put(complex(gamB1,phi),     8);
WFsteps.put(complex(gamB1,phibar), 9);
WFsteps.put(complex(gamB1,phi),   10);
WFsteps.put(complex(gamB1,phibar),11);
WFsteps.put(complex(gamB1,phi),    12);
WFsteps.put(complex(gamB1,phibar),13);
WFsteps.put(complex(gamB1,phi),    14);
WFsteps.put(complex(gamB1,phibar),15);
WFsteps.put(complex(gamB1,phi),    16);
WFsteps.put(complex(gamB1,phibar),17);
WFsteps.put(complex(gamB1,phi),    18);
WFsteps.put(complex(gamB1,phibar),19);
WFsteps.put(complex(gamB1,phi),    20);
WFsteps.put(complex(gamB1,phibar),21);
WFsteps.put(complex(gamB1,phi),    22);
WFsteps.put(complex(gamB1,phibar),23);
WFsteps.put(complex(gamB1,phi),    24);
```

Next we make a vector of step (pulse) lengths. These will be adjusted to produce the proper pulse angles for GARP:

```
row_vector WFtimes(25);                         // Vector for step times
double tdegree = 0;                             // Increment time per pulse
if(gamB1>0) tdegree = 1/(gamB1*360);            // degree
WFtimes.put(30.5*tdegree,  0);
WFtimes.put(55.2*tdegree,  1);
WFtimes.put(257.8*tdegree, 2);
WFtimes.put(268.3*tdegree, 3);
WFtimes.put(69.3*tdegree,  4);
WFtimes.put(62.2*tdegree,  5);
WFtimes.put(85.0*tdegree,  6);
WFtimes.put(91.8*tdegree,  7);
WFtimes.put(134.5*tdegree, 8);
WFtimes.put(256.1*tdegree, 9);
WFtimes.put(66.4*tdegree,  10);
WFtimes.put(45.9*tdegree,  11);
WFtimes.put(25.5*tdegree,  12);
WFtimes.put(72.7*tdegree,  13);
WFtimes.put(119.5*tdegree, 14);
WFtimes.put(138.2*tdegree, 15);
WFtimes.put(258.4*tdegree, 16);
WFtimes.put(64.9*tdegree,  17);
WFtimes.put(70.9*tdegree,  18);
WFtimes.put(77.2*tdegree,  19);
WFtimes.put(98.2*tdegree,  20);
WFtimes.put(133.6*tdegree, 21);
WFtimes.put(255.9*tdegree, 22);
WFtimes.put(65.6*tdegree,  23);
WFtimes.put(53.4*tdegree,  24);
```

Now we can make our waveform:

```
PulWaveform GWF(WFsteps, WFtimes, “GARP-1”);
```

Now that we have a waveform we can make composite pulses and pulse cycles which can be used as single steps, mixing steps, or during acquisition steps during NMR simulations. Remember, the waveform itself is just a container for shaped/composite pulse information. In itself it doesn’t do any calculations.

3.8.4 Pulse Waveform Utility

In GAMMA, pulse waveforms have little functionality. The class exists largely to serve higher level pulse entities such as composite pulses and pulse cycle. In effect, one builds the pulses required in NMR simulations from pulse waveforms. To illustrate this, consider a simulation in which CHIRP decoupling is desired during acquisitions. The basic CHIRP waveform is shown in the following figure.

Typical Chirp Sequence Amplitude and Phase

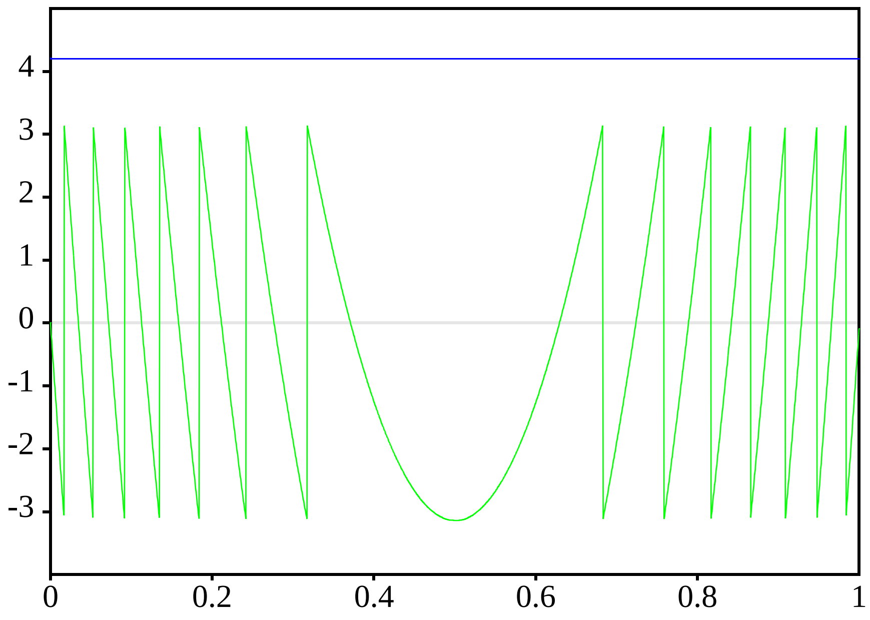

Figure 3-4 The rf amplitude and phase for a typical Chirp. The amplitide is kept constant at 4.2 kHz (scaled down by 1000 to fit on the plot) and ploted in blue. The phase is shown plotted in green and expressed in units of radians. The phase range is shown within the limits $phase \ni [-\pi,\pi]$. The program which produced the plot can be found in the GAMMA CHIRP documentation.

The previous section detailed how one might produce the above CHIRP waveform in GAMMA. To actually use a pulse based on this waveform ***demands*** more information than what is contained in the waveform itself. The computation must have knowledge of the spin system evolving, the channel the pulse will be applied on, possibly even dynamical and exchange parameters. By joining a spin system with a CHIRP pulse waveform we can produce a **composite pulse** that can indeed evolve the system under a CHIRP pulse. Here is some example GAMMA code:

```
spin_system sys;                          // Declare a spin system
sys.read("filein.sys");                   // Read in the spin system
PulWaveform WF = WF_CHIRP95(4200);        // Set up a CHIRP-95 waveform
PulComposite PC(WF, sys, "13C");          // Composite pulse on 13C channel
```

We could easily use repeated CHIRP composite pulses to evolve during a t1 evolution or during an acquisition. But, as is typical in using such decoupling sequences, the base waveform is phase cycled when repeatedly applied. In fact, CHIRP-95 is defined to be the base waveform cycled with a 5 step Tyko-Pines phase change and they supercycled in typical MLEV-16 fashion. Thus, if the CHIRP waveform is broken up into 200 steps the full CHIRP-95 decoupling sequence will be 16,000 steps long per each cycle. To perform a simulated acquisiton during such a process demands that the program is very careful in timing acquisition points relative to position in the cycle. In additon it is somewhat important to reutilize mathmatical entities which are repeatedly called for due to the nature of the sequence's symmetry. In GAMMA, these factors are automatically taken into account with "higher" pulse classes: PulCycle and PulSupCycle.

Relationship Between Chirp, Chirp Cycle, and Chirp Supercycle

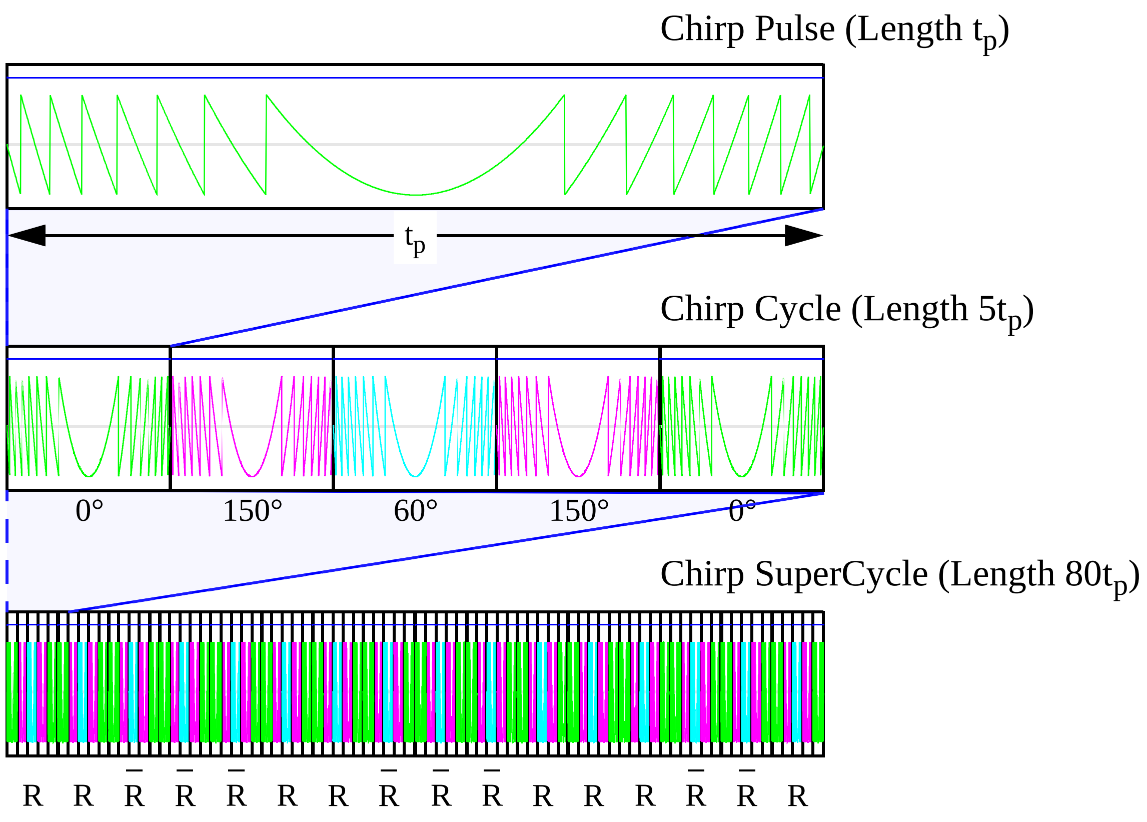
Chirp Pulse (Length t_p)
t_p
Chirp Cycle (Length $5t_p$)
0°
150°
60°
150°
0°
Chirp SuperCycle (Length $80t_p$)
R R $\bar{R}$ $\bar{R}$ $\bar{R}$ R R $\bar{R}$ $\bar{R}$ $\bar{R}$ R R R $\bar{R}$ $\bar{R}$ R

Figure 3-5 The relationship between the Chirp supercycle used for broadband decoupling and the basic Chirp sequence. The Chirp cycle is that developed by Tyko and Pines and consists 5 Chirps in succession with relative phase changes. The Chirp supercycle contains 16 of the Chirp 5-step cycles, each of which has a relative phase of zero (R) or 180 (R with a bar). The supercycle is MLEV-16 containing where the 5-step Chirp cycle is the primary unit. Thus there are 80 Chirp units in 1 supercycle, and the supercycle is repeated during decoupling.

For clarity lets summarize. Any ***waveform*** can be declared in GAMMA (e.g. CHIRP). To apply a pulse associated with the waveform one uses a ***composite pulse*** built from the waveform (or provided by a GAMMA function). If the pulse is to be repeated with a phase cycle one uses a ***pulse cycle*** made from the composite pulse. If the pulse cycle is itself cycled then one uses a ***pulse supercycle*** which is made from the pulse cycle.

3.9 Chapter Source Codes

GaussWF.cc

```
/* GaussWF.cc *********************************************************************
**                                                                              **
**                 GAMMA Pulse Waveform Example Program                         **
**                                                                              **
**  This program uses the class PulWaveform and the Gaussian pulse              **
**  module to build a Gaussian pulse waveform.  It doesn't do anything          **
**  with the waveform but spit out a waveform plot to the screen using          **
**  Gnuplot and make a FrameMaker MIF file of the plot.  The plot is            **
**  set for time vs rf-amplitude.                                               **
**                                                                              **
** Author:      S.A. Smith                                                      **
** Date:        3/9/98                                                          **
** Update:      3/9/98                                                          **
** Version:     3.5.4                                                           **
** Copyright:   S. Smith.  You can modify this program as you see fit          **
**              for personal use, but you must leave the program intact        **
**              if you re-distribute it.                                        **
**                                                                              **
***********************************************************************************/

#include <gamma.h>

main()
  {
  PulWaveform PW = WF_Gaussian(600, 0.01, 50);
  PW.GP(1, 5, 20);
  PW.FM(1, 5, 20);
  cout << "\n";
  cout.flush();
  }
```

GarpWF0.cc

```
/* GarpWF0.cc *********************************************************************  **
**                                                                                  **
**                    GAMMA GARP Simulation Example Program                         **
**                                                                                  **
** This program examines the basic GARP sequence Waveform.  I does no               **
** NMR computations involving GARP, it merely spits out plots so that               **
** the default GARP (GARP-1) sequence can be readily viewed.                        **
**                                                                                  **
** Assuming a.out is the executable of this program, then the following             **
** command will generate a single 25 step GARP-1 waveform which will                **
** be displayed on screen if Gnuplot is available.  It will also make               **
** an editable FrameMaker MIF file of the waveform.                                 **
**                                                                                  **
**                             a.out                                                **
**                                                                                  **
** Author:    S.A. Smith                                                            **
** Date:      2/27/98                                                               **
** Copyright: S.A. Smith, February 1998                                             **
**                                                                                  **
*************************************************************************************/

#include <gamma.h>                                     // Include GAMMA

main(int argc, char* argv[])
  {
  cout << "\n\n\t\t\tGAMMA GARP Waveform Program 0\n";
  GARP GP(500.0, "1H");                               // Set GARP parameters
  PulWaveform PWF = GP.WF_GARP1();                    // Construct waveform
  PWF.GP(1, 1, 5);                                    // Plot waveform(s), gnuplot
  PWF.FM(1, 1, 5);                                    // Plot waveform(s), Framemaker
  cout << "\n\n";                                     // Keep screen nice
  }
```